Module 10 Homework - Tie-breaking max heap

Part 1 - `class Entry`

Create a class `Entry` for priority queue entries that supports an arbitrary number of different priorities to break ties. Entries should be compared according to the first priority, with ties resolved by subsequent ones.

```
>>> e1 = Entry(priority=[0], item="jake")              # 1 level priority
>>> e2 = Entry(priority=[1], item="rachel")
>>> e2 > e1 # 1 > 0
True
>>> e1 = Entry(priority=[1, "a"], item="jake")          # 2 levels priority
>>> e2 = Entry(priority=[1, "b"], item="rachel")
>>> e3 = Entry(priority=[0, "c"], item="tobias")
>>> e2 > e1 # 1==1, 'b' > 'a'
True
>>> e2 > e3 # 1 > 0
True
>>> e1 = Entry(priority=[0, "a", 3.72], item="jake")     # 3 levels priority
>>> e2 = Entry(priority=[0, "a", 4.73], item="rachel")
>>> e2 > e1 # 0==0, 'a'=='a', 4.73 > 3.72
True
```

If a given level of priority is not specified for one of the entries, it should be treated as a minimum (i.e. the last object served by a max-heap):

```
>>> e1 = Entry(priority = [0])
>>> e2 = Entry(priority = [0, "a"])
>>> e2 > e1 # 0==0, 'a' > {Nothing}
True
```

Part 2 - `class MaxHeap`

The textbook and lectures introduced a binary min heap - the smallest priority was always at the top of the heap. Here, we will use a binary max heap, with the largest priority kept on top:

```
# A binary max heap
#      5
#    /   \
#   2     3
#  / \   / \
# 1   1 2   2
```

- Create a binary max heap `MaxHeap` that supports `put` and `remove_max`
- `put` should take an `Entry` type object as input - `def put(self, entry)`
- `remove_max` should return just the **item** associated with an entry
- Raise a RuntimeError if someone tries to remove from an empty heap

Test Guidelines

- See `TestMaxHeap.py` for unittest outlines. Feel free to add extra unittests.

Part 3 - Heapify

Add two methods for heapifying - `_heapify_up` and `_heapify_down`. These should treat `self._L` as a random list and sort it into a heap, using either `_upheap` or `_downheap` operations:

```python
def __init__(self, items=None, heapify_direction=None):
        self._L = []

        # if a collection of items is passed in, heapify it
        if items is not None:
            self._L = list(items)

            if heapify_direction == 'up': self._heapify_up()

            elif heapify_direction == 'down': self._heapify_down()

            else:
                raise RuntimeError("Replace `heapify_direction` default with \
                    'up' or 'down' instead of `None`")

    def _heapify_up(self):
        """Heapifies self._L in-place using only upheap"""

    def _heapify_down(self):
        """Heapifies self._L in-place using only downheap"""
```

`test_heapify()` (provided with unitest skeleton code) times each method. Replace the default value for `heapify_direction` in `MaxHeap.__init__()` with either `'up'` or `'down'`, whichever is faster.

```
n       t_h_up (ms)     t_h_down (ms)
1000    ???             ???
2000    ???             ???
3000    ???             ???
4000    ???             ???
```

Submitting

At a minimum, submit the following file:

- `MaxHeap.py`
- `TestMaxHeap.py`

Students must submit individually by the due date (**Tuesday 11/29** at 11:59 pm EST) to receive credit.

Grading

This assignment is 100% manually graded.

Feedback

If you have any feedback on this assignment, please leave it here.